Cascading style sheets

1. CSS styling

a. In html

i. External CSS

External file in imported in html

```
<head>
  <link rel="stylesheet" href="mystyle.css">
</head>
```

ii. Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.

```
<!DOCTYPE html>
<html>
<head>
 <style>
   body {
      background-color: linen;
   }

   h1 {
      color: maroon;
      margin-left: 40px;
    }
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

iii. Inline CSS

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;text-align:center;">This is a
heading</h1>
<p style="color:red;">This is a paragraph.</p>

</body>
</html>
```

b. In react

i. Inline styling
ii. CSS Modules
iii. SASS

2. CSS Preprocessor

a. CSS preprocessors are scripting languages that extend the capabilities of CSS, allowing developers to write CSS in a more dynamic and efficient way. They enable you to use features like variables, nesting, mixins, functions, and more, which are not available in standard CSS.

b. Some of them are SASS (Syntactically Awesome Stylesheet), LESS, and STYLUS.(.scss for SASS, .less for LESS)

c. CSS preprocessors work by taking the code you write in their specific syntax, preprocessing it into standard CSS, compiling it into a single file, optimizing the generated CSS, and outputting it.

d. **Adv**

i. **Easier to maintain.**

For example, you declare a variable for your primary color and secondary color:

```
$primary_color: #346699;
$secondary_color: #769bc0;
```

And then you can use it like this:

```
a{color: $primary_color;}
nav{background-color: $secondary-color;}
```

Which will output the CSS as:

```
a{color: #346699;}
nav{background-color:#769bc0;}
```

ii. **<u>It will make your CSS DRY (don't repeat yourself)</u>**

How often have you written something like this in CSS?

```
.main-heading {
    font-family: Tahoma, Geneva, sans-serif;
    font-weight: bold; font-size:20px; text-
    transform: uppercase; color: blue;

}


.secondary-heading {

    font-family: Tahoma, Geneva, sans-serif;
    font-weight: bold; font-size:16px; text-
    transform: uppercase; color: blue;

}
```

Written in SCSS:

```
.main-heading {
    font-family: Tahoma, Geneva, sans-serif;
    font-weight: bold; font-size:20px; text-
    transform: uppercase; color: blue;

}
```

```
.secondary-heading{
        @extend .main-heading; font-size:16px;

}
```

Using @extend in SCSS will share a set of CSS properties from one select to another. It helps keep your SCSS very DRY.

iii. More Organized

1. Support nested feature.

```
h2 a {
        color: blue;

}

h2 a:hover {
        text-decoration: underline;

        color: green;

}
```

3. Simple CSS Selectors:

a. Element selector:

```
p {
  text-align: center;
  color: red;
}
```

b. Id selector:

The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

c. Class selector:

In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {
  text-align: center;
  color: red;
}
```

d. Universal selector:

```
* {
  text-align: center;
  color: blue;
}
```

e. Group selector:

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

4. Other Selectors:

a. Combinator Selectors:

i. Descendent selector:

Select all the descendent element even nested element.

```
div p {
  background-color: yellow;
}
```

ii. Child selector:

The child selector selects all elements that are the children of a specified element but not the nested element.
Selects all <p> elements where the parent is a element

```
div > p {
  background-color: yellow;
}
```

iii. Adjacent sibling selector:

The adjacent sibling selector is used to select an element that is directly after another specific element.
Selects the first <p> element that are placed immediately after elements

```
div + p {
  background-color: yellow;
}
```

just after the div element not its children.

iv. General sibling selector:

```
div ~ p {
  background-color: yellow;
}
```

Selects every <ul> element that are preceded by a <p> element

b. Pseudo Classes:

i. Anchor Pseudo class

1. a:link { } // select all unvisited links
2. a:visited { }. // select all visited links
3. a:hover { }
4. a:active { } // select active link

a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective! a:active MUST come after a:hover in the CSS definition in order to be effective! Pseudo-class names are not case-sensitive.

ii. Other pseudo classes

1. p:first-child { }
2. div:not(p) // select div every element which is not <p>
3. :focus
4. :last-child
5. :nth-child(n)
6. :nth-last-child(n)

iii. Pseudo elements:

1. p::after // insert content after every p element
2. p::before. //insert content before every p element

5. CSS Architecture:

a. CSS architecture refers to the organization and structuring of CSS code for better maintainability, scalability, and efficiency. There are various CSS architectures available, but one of the most popular ones is **BEM (Block Element Modifier).**

BEM is a methodology that uses a specific naming convention to name classes in a consistent and modular way. It uses three types of entities:

Blocks: standalone entities that represent a significant part of the UI like container for instance,

Elements: parts of a block that have no meaning outside of the context of the block like <p>, <h1> within the container

like . for example “container__p” naming <p> tag within container.

Modifiers: variations of a block or element that change their appearance or behavior (e.g., disabled, active ,colortheme) like container__p—active, container__p—disabled

Here's an example of how to use BEM:

```
/* Block */
.menu { display: flex; justify-content: space-between; }

/* Elements */
.menu__item { list-style: none; }

/* Modifier */
.menu__link--active { font-weight: bold; }
```

6. CSS Display:

a. Block-level element

i. Default style

1. A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

Examples of block-level elements:

, <h1> - <h6>, <p>, <form>, <header>, <footer>, <section>

ii. Overridden by display: inline;

b. Inline Element

i. An inline element does not start on a new line and only takes up as much width as necessary.

Examples of inline elements: , <a>, <img>

ii. Can be overridden by using display: block;

c. Inline-block:

i. Compare to inline → inline-block able to put width, height and top/bottom margin/padding are respected.
ii. Compare to block →inline-block does not allow line break.

d. None:

i. Hide specific element that you don't want to show.
ii. display: none;

7. CSS Position and Z-index:

a. Static:
   i. Remain as normal, no changes
   ii. Can't use top, right, bottom..
   iii. position: static;

b. Relative:
   i. Positioned in normal flow.
   ii. Can use top, right.. to arrange element as desire.
   iii. position: relative;

c. Fixed:
   i. Remain at the same place, even page is scrolled.
   ii. position: fixed;

d. Absolute:
   i. If any element is position: absolute, then that element space will not be allocated and stick with page scroll.
   ii. If parent container is position: relative, and child element is position: absolute, then the child element stick to the parent element only.

e. Sticky
   i. Toggle between relative and fixed, totally depend upon user's scroll behaviour.

f. **Z-index:**

i. The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others). An element can have a positive or negative stack order.

ii. z-index only works on positioned elements (position: absolute, position: relative, position: fixed, or position: sticky) and flex items (elements that are direct children of display: flex elements).

8. Other CSS property:

a. Width, Max-Width, Margin

i. a block-level element takes full width available as it can. Setting the width of a block-level element will prevent it from stretching out to the edges of its container. Then, you can set the margins to auto, to horizontally center the element within its container.

ii. The element will take up the specified width, but the problem with the above occurs when the browser window is smaller than the width of the element. The browser then adds a horizontal scrollbar to the page.

iii. Using max-width instead, in this situation, will improve the browser's handling of small windows. This is important when making a site usable on small devices.

iv. using max-width causes the element's width can be less than or equal to the max-width value, but never greater.

b. Box Sizing:-

i. By default, the width and height of an element is calculated like this:

ii. width + padding + border = actual width of an element
height + padding + border = actual height of an element

iii. This means: When you set the width/height of an element, the element often appears bigger than you have set (because the element's border and padding are added to the element's specified width/height). Like if one box padding is used, then the other box width and height squeezed.

iv. The box-sizing property allows us to include the padding and border in an element's total width and height, without altering or merging the each other width and height.

v. If you set box-sizing: border-box; on an element, padding and border are included in the width and height:

c. Media Query:-

i.
```
@media screen and (min-width: 480px) {
  #leftsidebar {width: 200px; float: left;}
  #main {margin-left: 216px;}
}
```

d. Overflow.
e. Align Element using position, float.
f. Opacity
g. Border-radius
h. Box model
i. Text effects
j. Linear, radial gradient
k. Outline, text, fonts, icons, background, background-images, color keyword,
l. Object-position
m. CSS masking
n. Css animation → later on

o. RGBA()

i. RGBA(red, green, blue, opacity)
   1. Each parameter define the intensity between 0 to 255.

ii. Hex:
   1. #rrggbb →specified using hexadecimal value.
   2. Determine between 00 to FF, black is #000000, white is # ffffff

p. Specificity

i. Hierarchy of which style will be implemented , if implemented at the same time.

Inline styles - Example: <h1 style="color: pink;">

IDs - Example: #navbar

Classes, pseudo-classes, attribute selectors - Example: .test, :hover, [href]

Elements and pseudo-elements - Example: h1, ::before

ii. !important : - it will override ALL previous styling rules for that specific property on that element!

q. Math functions:-

i. calc()
ii. max()
iii. min()

r. Units:-

i. % :- relative to parent element

ii. px :- 1/96th of 1 inch
iii. em :- Relative to the font-size of the element (2em means 2 times the size of the current font)
iv. rem :- relative to font size of the root element
v. vw :- relative to 1% of width of viewport
vi. vh :- relative to 1% of height of viewport

9. Box Shadow:-

a. Syntax:- box-shadow: [horizontal offset] [vertical offset] [blur radius] [optional spread radius] [color];

   i. box-shadow: 0 3px 10px 0 rgba(0,0,0,0.2);

b. use inset keyword for inner shadow
   i. box-shadow: 10px 10px 5px lightblue inset;

c. multiple shadow
   i. box-shadow: 0 4px 8px 0 rgba(0,0,0,.2),0 6px 20px 0 rgba(0,0,0,0.19)

10. CSS transformation property:

a. **transform:**- Applies a 2D or 3D transformation to an element

   All the following functions are applied for 2D effects. **There are function for 3D effects refer w3school.**

   i. matrix(n,n,n,n,n,n) Defines a 2D transformation, using a matrix of six values
   ii. translate(x,y) Defines a 2D translation, moving the element along the X- and the Y-axis
   iii. translateX(n) Defines a 2D translation, moving the element along the X-axis
   iv. translateY(n) → Defines a 2D translation, moving the element along the Y-axis

v. scale(x,y) → Defines a 2D scale transformation, changing the elements width and height
vi. scaleX(n)→ Defines a 2D scale transformation, changing the element's width
vii. scaleY(n)→ Defines a 2D scale transformation, changing the element's height
viii. rotate(angle)→ Defines a 2D rotation, the angle is specified in the parameter
ix. skew(x-angle,y-angle)→Defines a 2D skew transformation along the X- and the Y-axis
x. skewX(angle)→Defines a 2D skew transformation along the X-axis
xi. skewY(angle)→Defines a 2D skew transformation along the Y-axis

11. CSS translation property:

a. Kati ko gati ley translation garney vhaney ko ho.

b. **transition:** [transition-property] [transition-duration] [transition-timing-function] [transition-delay];

c. transition-timing-function:

i. ease
ii. linear
iii. ease-in
iv. ease-out
v. ease-in-out
vi. cubic-bezier(n,n,n,n)

12. object-fit:

a. object-fit property is used to specify how an <img> or <video> should be resized to fit its container.

i. fill - This is default. The image is resized to fill the given dimension. If necessary, the image will be stretched or squished to fit
ii. contain - The image keeps its aspect ratio, but is resized to fit within the given dimension
iii. cover - The image keeps its aspect ratio and fills the given dimension. The image will be clipped to fit
iv. none - The image is not resized
v. scale-down - the image is scaled down to the smallest version of none or contain

13. Flex Box:-

a. display : flex;

b. flex-direction

i. flex-direction: row | row-reverse | column | column-reverse;

c. flex-wrap:

i. flex-wrap: nowrap | wrap | wrap-reverse;
ii. nowrap(default) –> all flex items will be on one line.
iii. wrap → flex items wrap on multiple lines, from top to bottom.
iv. wrap-reverse → flex-items will wrap onto multiple lines from bottom to top.

d. justify-content:
i. center
ii. flex-start
iii. flex-end
iv. space-between
v. space-around
vi. space-evenly

e. align-items:
    i. flex-start
    ii. flex-end
    iii. center
    iv. stretch → stretch to fill the container (still respect min-width/max-width)
    v. baseline → items are aligned such as their baseline

    Default(row) → justify-content align the items horizontally and align-items align the items vertically.

    On flex-direction: column → justify-content and align-items role switch up.

f. align-content:
    i. Individual items within a container are aligned with align-items, whereas a set of things within a container is aligned with align-content.
    ii. All the properties are same.
    iii. Can be used only when flex-wrap: wrap;
    iv. Used to align the flex line.

g. gap:
    i. gap is a shorthand property for both row-gap and column-gap that indicates the size of the gap between grid or flexbox items' rows and columns. Gap: 20px, for example, would generate a 20-pixel gap between rows and columns.
    ii. row-gap: 10px, column-gap:20px; each can be used to determine the gap individually.

h. order:

i. The order property specifies the order of the flex items.

ii. The first flex item in the code does not have to appear as the first item in the layout.

iii. The order value must be a number, default value is 0.

iv.
```
<div class="flex-container">
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
  <div style="order: 1">4</div>
</div>
```

i. flex:

i. shorthand property for flex-grow, flex-shrink, flex-basis.

ii. Syntax → flex: flex-grow, flex-shrink, flex-basis(initial length)

14. CSS Grid:

a. ***Container***: property should be listed in container.

i. **display**

1. gird → generate block level grid.
2. inline-grid → inline level grid.

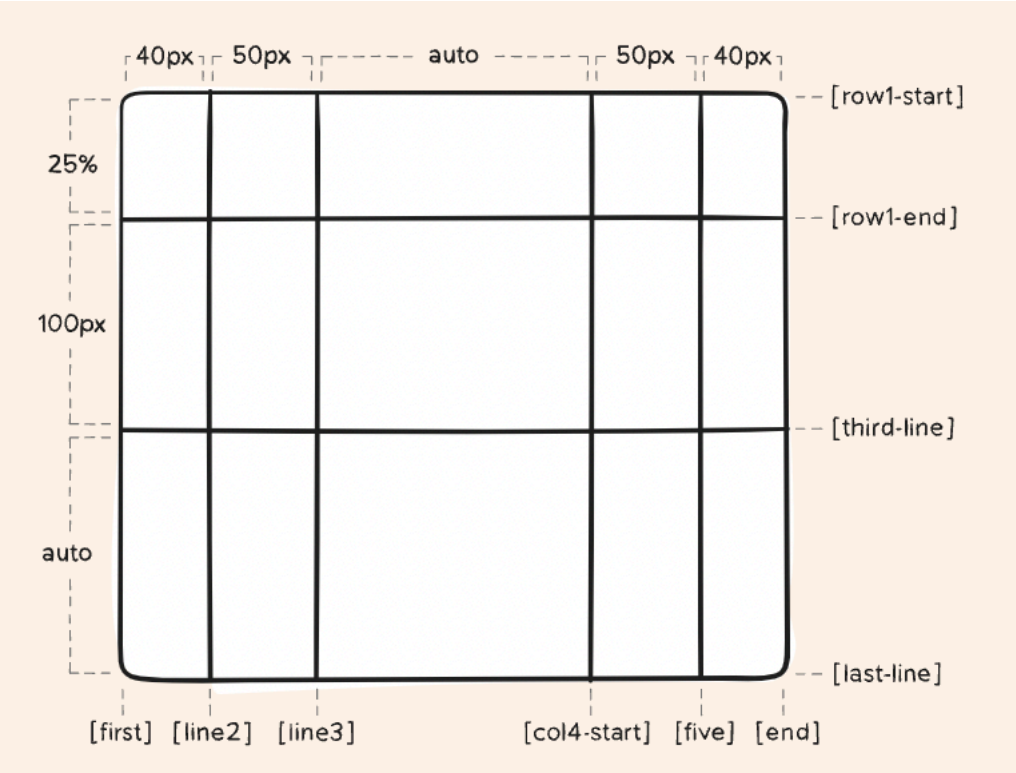
ii. **grid-template-columns | rows**

1. track size → length, percentage, fraction(fr)
2. line-name → choose your self

```
.container {

    grid-template-columns: [first] 40px [line2]
        50px [line3] auto [col4-start] 50px
        [five] 40px [end];
```

```
        grid-template-rows: [row1-start] 25%
                [row1-end] 100px [third-line] auto
                [last-line];
}
```



3. You can explicitly name each line like in [ ] above that is line-name and numbers are track-size.
4. Can have multiple name inside [ ]
5. If it consist repeating parts then use repeat() like, repeat(3, 20px, [col-start]) → times repeated, size, name.

iii. **grid-template-areas:**

1. gird-area-name → name of the grid area specified with grid-area.
2. **‘.’** → period signifies an empty grid cell.
3. None → no gird cell are defined

Examples

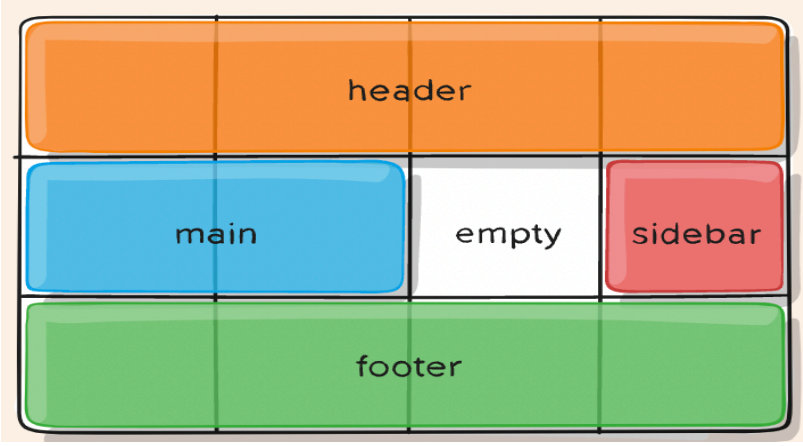
```
.item-a {
  grid-area: header;
}
.item-b {
  grid-area: main;
}
.item-c {
  grid-area: sidebar;
}
.item-d {
  grid-area: footer;
}

.container {

  display: grid;
  grid-template-columns: 50px 50px
                                50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
        "header header header header"
         "main main . sidebar"
       "footer footer footer footer";

}
```

→Each time name repeated it takes the column space like header.



iv. **grid-template:**
   1. shorthand for grid-template-columns, grid-template-rows, grid-template-areas.
   2. Refer CSS tricks.

v. **gap:-**
   1. short hand for column-gap and row-gap
   2. Syntax → grid: <grid-row-gap> <grid-column-gap>

vi. **Aligning items:-**

→ value applies to all the grid items inside the container.

   1. **Justify-items:**
      a. start, end, center, stretch.
      b. Use horizontal axis to align.

   2. **align-items:**
      a. start, end, center, stretch.
      b. Align using vertical axis.

vii. **place-items:**
   1. The place-items property in CSS grid is used to align items in both the horizontal

and vertical directions. It is a shorthand property that combines the align-items and justify-items properties.

2. **Examples:**
    a. place-items: center;
    b. place-items: start;

viii. **Aligning Content:**

→ applied to the whole content, values applied to this property are same as above.

1. justify-content
2. align-content
3. place-content

ix. **Extra:**

1. gird-auto-columns
2. gird-auto-rows
3. gird-auto-flow
4. grid

b. **Children:-**

i. **Determining grid location:**

1. .item {
    grid-column-start: <number> | <name> | span <number> | span <name> | auto;

    grid-column-end: <number> | <name> | span <number> | span <name> | auto;

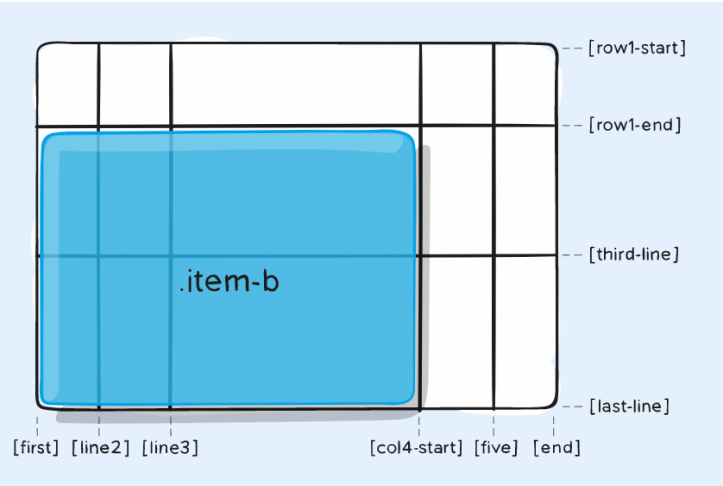
    grid-row-start: <number> | <name> | span <number> | span <name> | auto;

    grid-row-end: <number> | <name> | span <number> | span <name> | auto;
}

- <line> – can be a number to refer to a numbered grid line, or a name to refer to a named grid line span
- <number> – the item will span until it hits next line with the provided number span
- <name> – the item will span across until it hits the next line with the provided name
- auto – indicates auto-placement, an automatic span, or a default span of one

Examples

```
.item-b {
  grid-column-start: 1;
  grid-column-end: span col4-start;
  grid-row-start: 2;
  grid-row-end: span 2;
}
```



ii. **Shorthand for above:**

.item {

grid-column: <start-line> / <end-line> | <start-line> / span <value>;

grid-row: <start-line> / <end-line> | <start-line> / span <value>;

}

c. **gird-area:**

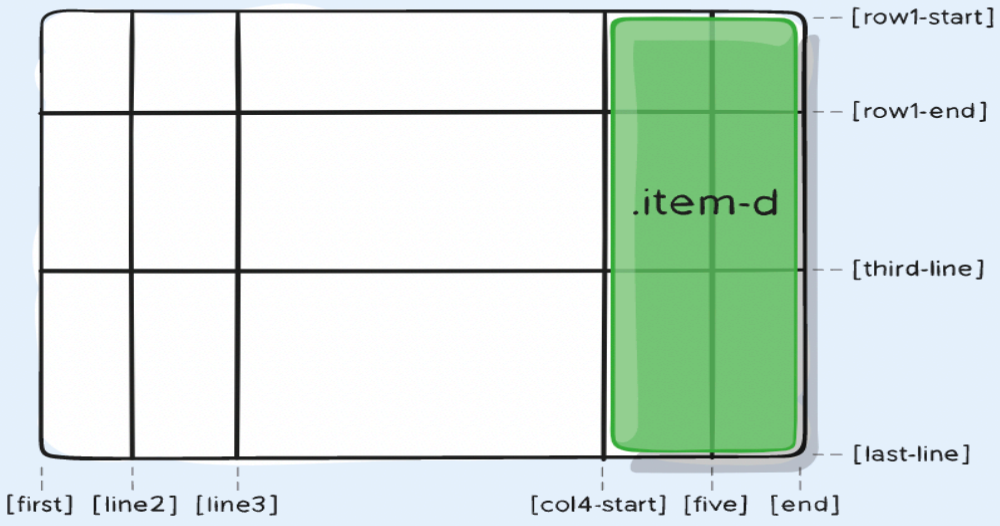
i. referenced by grid-template-area in container.

.item {

grid-area: <name> | <row-start> / <column-start> / <row-end> / <columnend>;

}

```
.item-d {
  grid-area: 1 / col4-start / last-line / 6;
}
```



```
item{
        grid-area: header;  // naming
}
```

d. aligning the items inside cell
    i. justify-self
    ii. align-self
    iii. place-self